



**Stiftung University of Hildesheim**  
Marienburger Platz 22  
31141 Hildesheim  
Germany



**Software Systems Engineering (SSE)**  
Institute for Computer Science  
Faculty for Mathematics, Natural  
Science, Economics, and Computer  
Science



# **EASy-Producer**

Engineering Adaptive Systems

## **User Guide**

Version 1.1

11.07.2014

## **Version**

0.1	23.08.2012	Initial version.
0.2	10.09.2012	Table of content, initial introduction, prerequisites, and installation section added.
0.3	22.10.2012	Changes due to migration to Xtext version 2.3.1, preface added, modification and extension of Sections 1, 3 and 4. Sections 2, 5, and appendix initially added.
0.4	30.12.2012	Section 5 updated (screenshots and descriptions added).
0.5	04.03.2013	Section 3 updated (inclusion of Xtext features in EASy update site).
0.6	02.09.2013	Section 4 updated (inclusion of VIL)
1.0	27.09.2013	Version 1.0 completed (reference to other developers guide, IVML and VIL language specification, corrected spelling, updated figures)
1.1	11.07.2014	FAQ (Section 6) added.

### Preface

EASy-Producer is a Software Product Line Engineering tool developed by the Software Systems Engineering (SSE) group at the University of Hildesheim.

The tool is available as an Eclipse plug-in under the terms of the Eclipse Public License (EPL) Version 1.0

The SSE group hosts the following EASy-Producer update site for easy installation and updates:

<http://projects.sse.uni-hildesheim.de/easy/>

## Table of Contents

1.	Introduction.....	6
2.	Software Product Line Engineering at a Glance .....	7
2.1.	Basic Software Product Line Engineering.....	7
2.2.	Staged Configuration and Instantiation.....	7
2.3.	Multi Software Product Lines .....	7
3.	Installation .....	9
3.1.	Prerequisites.....	9
3.2.	Installation: Step by Step .....	9
3.3.	Technical Recommendations.....	11
3.4.	Further Guides and Specifications.....	11
4.	Getting Started: Product Line Engineering is EASy .....	13
4.1.	Running Example.....	13
4.2.	Defining a New Base Service Platform.....	14
4.2.1.	Configuration Space Definition .....	15
4.2.2.	Implementation Space Definition.....	17
4.3.	Deriving a Domain-Specific Service Platform.....	19
4.3.1.	Configuration of a Domain-Specific Service Platform .....	19
4.3.2.	Instantiation of a Domain-Specific Service Platform.....	20
5.	EASy-Producer in Detail.....	22
5.1.	The Product Line Project Structure .....	22
5.2.	The Product Line Editor .....	22
5.2.1.	The Project Configuration Editor.....	23
5.2.2.	The IVML Configuration Editor .....	24
6.	Frequently Asked Questions (FAQ).....	26
6.1.	VIL-Editor won't work after updating EASy/Xtext.....	26
A.	Appendix.....	28
A.1.	Running Example IVML-File .....	28
A.2.	Running Example VIL Build Script-File.....	29



## **1. Introduction**

EASy-Producer<sup>1</sup> is a Software Product Line Engineering (SPLE) tool which facilitates the most recent trends and concepts in SPLE, such as large-scale Multi-Software Product Lines (MSPL), product line hierarchies, and staged configuration and instantiation. The focus of this tool is to support these rather complex concepts in an easy-to-use way. Thus, this tool allows developing a first prototypical Software Product Line (SPL) within minutes. Further, EASy-Producer is a research prototype for demonstrating new approaches to SPLE in general and, in particular approaches for simplifying the development of SPLs developed by the Software Systems Engineering group (SSE) at the University of Hildesheim.

This live-document provides a user guide that introduces the reader to the concepts and capabilities of EASy-Producer. In Section 2, we will give a brief overview on the SPLE concepts supported by EASy-Producer. This will include introductions to the concepts of SPLE in general, staged configuration and instantiation, MSPL, and product line hierarchies.

Section 3 will give guidance for the first steps in EASy-Producer. This section includes the mandatory prerequisites, the installation guide, and additional recommendations for running the tool successfully.

In Section 4, we will introduce EASy-Producer in terms of describing the development of a first prototypical SPL and the derivation of a product line product. This will cover all aspects of SPL development ranging from creating a new product line project in EASy-Producer, defining a variability model and implementing the corresponding product line artefacts, to the derivation, configuration, and instantiation of a specific product. While the purpose of this section is to describe and illustrate the basic application of EASy-Producer, we will not discuss all details of the tool at this point. This will be part of the next section.

Finally, Section 5 will describe EASy-Producer in detail. This includes detailed descriptions of the individual editors and views of the tool.

---

<sup>1</sup> EASy is an abbreviation for Engineering Adaptive Systems.

## 2. Software Product Line Engineering at a Glance

EASy-Producer supports basic Product Line Engineering and also staged configuration and Multi Software Product Lines or any combination of these techniques. In the next three sections we will give a short introduction to these concepts.

### 2.1. Basic Software Product Line Engineering

Software Product Line Engineering (SPLE) is a software development approach which focuses on the extensive reuse of artefacts involved or produced in the software lifecycle. The overall goal of SPLE is to provide a high degree of automation for the configuration and adaptation of product variants. This approach reduces the development effort and costs as well as the time-to-market while increasing the overall software quality.

A Software Product Line (SPL) is a set of related software products which are developed based on a common infrastructure but differ with respect to their provided functionalities. These differences are called variabilities.

### 2.2. Staged Configuration and Instantiation

Staged configuration and especially staged instantiation are approaches for facilitating partial derivation of product artefacts. These partial instantiated artefacts can still contain open variabilities while other variabilities are already bound. Thus, the configuration can be connected in arranged series. This technique can be used to support different stakeholder/user groups or to create a common basis for related sub-sets of a product line. See Figure 1 for an illustrative example.

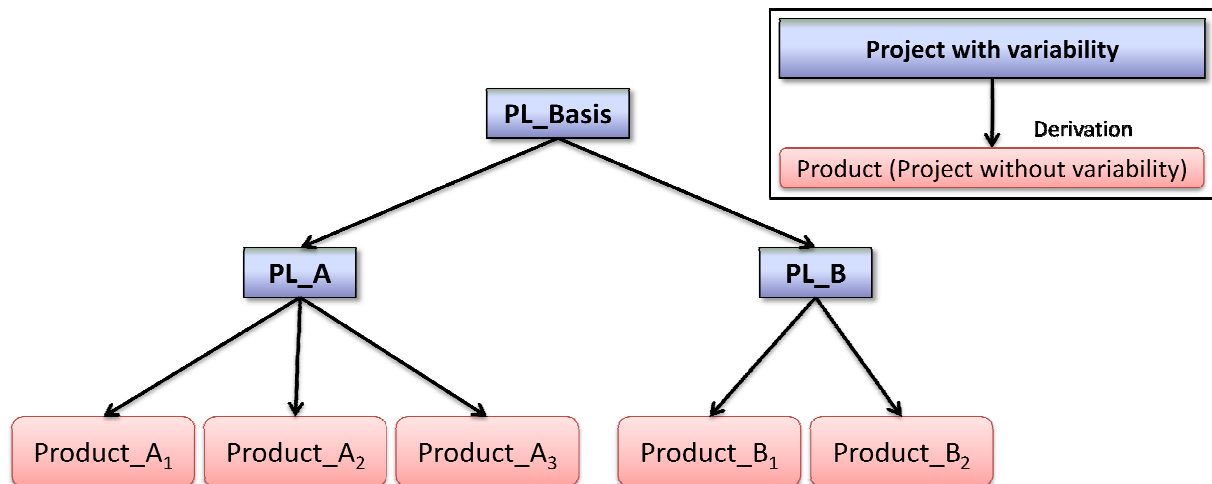


Figure 1: Example for staged configuration

### 2.3. Multi Software Product Lines

Multi Software Product Lines (MSPLs) are able to compose several (independent) product lines to form new products (or product lines). While forming an MSPL, the variability models of the single

product lines are combined to an integrated variability model. Derived products can contain instantiated artefacts from all combined product lines. See Figure 2 for an illustrative example.

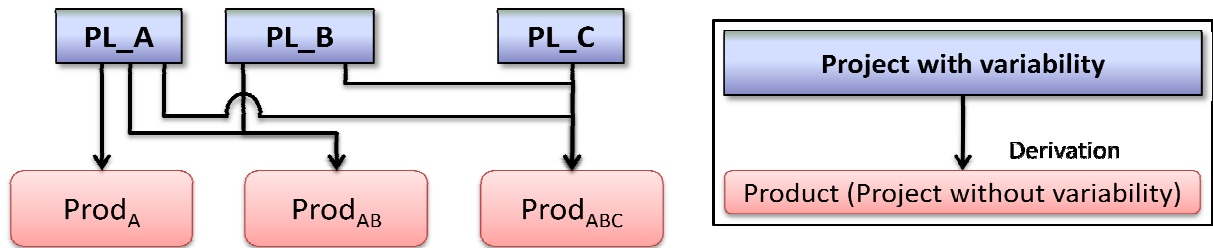


Figure 2: Example for a Multi Software Product Line.



## 3. Installation

In this section, we will describe the installation of EASy-Producer. In order to guarantee a successful installation, we will introduce a set of mandatory prerequisites. This will be part of Section 3.1 in which we will set up the environment in for EASy-Producer. In Section 3.2, we will describe the installation of the tool in a step-wise manner using the Eclipse update site mechanism and the EASy-Producer update site. Finally, Section 3.3 will give some technical recommendations, while Section 3.4 introduces additional guides and specifications for EASy-Producer.

### 3.1. Prerequisites

EASy-Producer is developed as an **Eclipse**<sup>2</sup> plug-in and requires **Xtext**<sup>3</sup> **version 2.3.1**. Thus, in general, any Eclipse installation with Xtext version 2.3.1 is fine for installing and running EASy-Producer. However, we cannot guarantee that any combination of Eclipse and Xtext version 2.3.1 will work with EASy-Producer. Thus, we propose the following Eclipse versions as they are tested with EASy-Producer (and Xtext version 2.3.1):

- Eclipse 3.6 (Helios)
- Eclipse 3.7 (Indigo)
- Eclipse 4.0 (Juno)

We recommend using **Eclipse 3.7 (Indigo)** as this is the most exhaustively tested version of Eclipse with EASy-Producer. Download an Eclipse package from <http://www.eclipse.org/downloads/>.

**Please note that Eclipse 4.2 does not work with Xtext 2.3.1 due to incompatible dependencies.**

Further, Xtext version 2.3.1 has to be installed in the newly downloaded Eclipse instance. Typically, this is installed automatically when installing EASy-Producer due to plug-in dependencies. However, we encountered situations in which these dependencies were not automatically resolved. Thus, the EASy-Producer update site includes the required Xtext features. We will describe the complete installation in the next Section.

### 3.2. Installation: Step by Step

The SSE group hosts an EASy-Producer update site for easy installation and updates. Thus, the first step for installing EASy-Producer is to define a new update site in Eclipse. For this purpose, start Eclipse and open the *Install New Software* dialog by clicking *Help* → *Install New Software...* as shown in Figure 3:

---

<sup>2</sup> Eclipse website: [www.eclipse.org/](http://www.eclipse.org/)

<sup>3</sup> Xtext website: <http://www.eclipse.org/Xtext/>

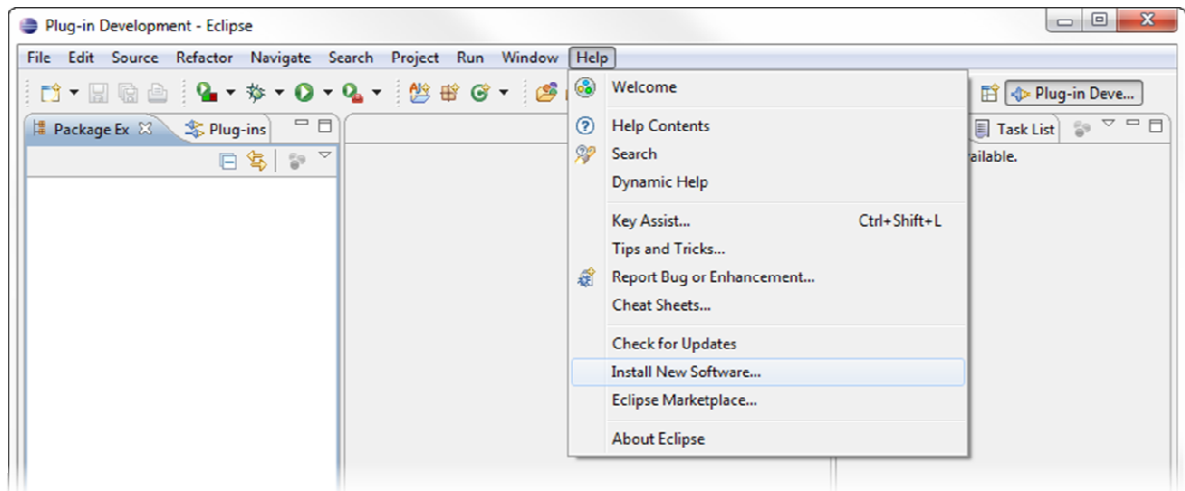


Figure 3: Open the “Install New Software” dialog

The *Install* Dialog will appear (cf. Figure 4). In this dialog, a new location for available software has to be added. Thus, click on the *Add...* button in the upper right location of the dialog.

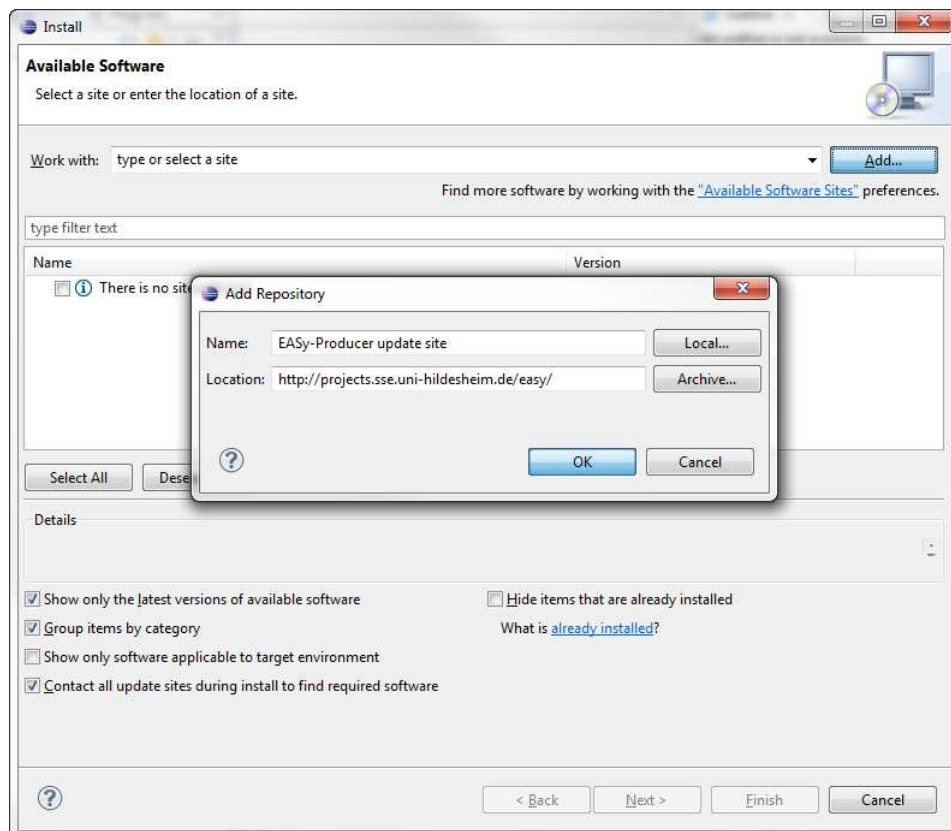


Figure 4: Add a new location for software updates

The *Add Repository* dialog requires the definition of a name for the new update site and a location as illustrated in Figure 4. The name is up to the user. For example, enter “EASy-Producer update site”. The location is the URL of the update site:

EASy-Producer update site: <http://projects.sse.uni-hildesheim.de/easy/>

Finish the definition of the new update site by clicking the OK button of the *Add Repository* dialog.

The *Install* Dialog will now contain multiple categories. If you are installing EASy-Producer for the first time and do not know which features to select, select the *Quick Installation of EASy-Producer* category. Further, select the categories *Xtend-2.3.1* and *Xtext-2.3.1* to install the required Xtext version (if not done before). This will install all required components automatically.

For more experienced users, select the categories and features as needed and click the *Next* button. Follow the steps for installing EASy-Producer (accept the license agreement and ignore the security warning for installing software of unsigned content, etc.), and restart Eclipse as prompted.

Finally, you have successfully installed the EASy-Producer.

### **3.3. Technical Recommendations**

In order to avoid memory problems while using EASy-Producer, we recommend increasing the memory of the Eclipse application in which EASy-Producer is executed. The memory problems are due to Xtext which requires more memory than defined in a typical Eclipse configuration.

Open the “*eclipse.ini*” file in your Eclipse directory and enter the following parameters at the end of the file:

```
-vmargs  
-Xms40m  
-Xmx512m  
-XXMaxPermSize=128m
```

### **3.4. Further Guides and Specifications**

EASy-Producer provides two expressive languages that support the creation of required software product line artefacts:

The **INDENICA Variability Modelling Language (IVML)** is an expressive, textual variability modelling language, which provides basic and advanced modelling capabilities for the definition of variability models. In order to define such a model based on IVML, we provide the IVML language specification. This specification is part of the EASy-Producer installation and can be found in the **Eclipse Help**.

The **Variability Implementation Language (VIL)** is a textual language for the flexible specification of the instantiation process of a software product line. This language consists (beside other parts) of the VIL build language and the VIL template language. The former language provides modelling elements for the specification of the individual build tasks of the instantiation process, while the latter language supports the definition of templates that can be applied to specific artefacts, for example, to manipulate their content, as part of the instantiation process. The corresponding VIL language specification is also part of the EASy-Producer installation and can be found in the **Eclipse Help**.

Further, EASy-Producer supports the extension of the tool by custom instantiations and reasoners. The **EASy-Producer Developers Guide** introduces the reader to the possible extensions and provides a step-wise description of how to extend the tool. This guide can be found in the **Eclipse Help** as well.

The EASy-Producer user guide, the EASy-Producer developers guide, as well as the IVML and the VIL language specification are also available as PDFs on the EASy-Producer update site.

## 4. Getting Started: Product Line Engineering is EASy

In this section, we will adopt the roles of a domain engineer and an application engineer in order to illustrate the application of EASy-Producer based on a running example. We will prototypically model and implement the variability of a content-sharing platform, which allows the user to upload, annotate, release and share content of various types. Section 4.1 will describe this example in detail. In Section 4.2, we will adopt the role of a domain engineer and describe the definition of a SPL from which multiple variants of the content-sharing platform can be derived. This includes the definition of the variability model using the INDENICA<sup>4</sup> Variability Modelling Language (IVML), the implementation of these variabilities in source code, and the definition of a build script for the instantiation of the generic artefacts using the Variability Implementation Language (VIL). In Section 4.3, we will adopt the role of an application engineer and describe the derivation of a specific service platform variant including the variant configuration and the instantiation of the corresponding artefacts.

We will use the following font styles throughout this section to illustrate and distinguish between actions, active tool elements, and added input:

- EASy-Producer (as well as Eclipse) provides multiple editors, wizards, etc. In order to identify the **active tool element** currently in use, it will be highlighted using bold font.
- All *actions* that will be performed will be highlighted using italics font.
- All input to EASy-Producer will be illustrated in `Courier New`.

Please note that we will not discuss the tool in all details in this section. This will be part of the detailed description of EASy-Producer in Section 5. Further, we will not discuss the IVML and VIL language here. A detailed description of these languages can be found in the corresponding language guides (cf. Section 3.4).

### 4.1. Running Example

In this section, we introduce a running example which we will use throughout Section 4 to illustrate the basic application and capabilities of EASy-Producer. In this example a content-sharing platform will be developed in terms of a SPL. A content-sharing platform allows its users to upload, annotate, release and share content of various types. In this example, concrete applications may differ with respect to:

- The supported content types such as text, video, audio, 3D content, or binary (large) objects (BLOBs).
- The hosting infrastructure which consists of a) a web container being responsible for serving the content and b) the database, which stores user and content data.
- The deployment target which may either be a traditionally hosted server, or a cloud environment. The cloud environment may be private, like a local installation of the

---

<sup>4</sup> INDENICA is an EU-funded project in which the variability modeling language of EASy-Producer was initially designed and developed. However, this language is not INDENICA-specific but was designed with further requirements from research and industry in mind. For more information regarding INDENICA please visit the INDENICA website: <http://indenica.eu/>

Eucalyptus<sup>5</sup> cloud software or public, in this example we will allow connections to Amazon<sup>6</sup> or Azure<sup>7</sup> cloud.

Without going into functional details of the content-sharing platform, the variabilities introduced by content types, web container, database and deployment target allow to derive a large number of different platform instances. However, some dependencies exist that restrict the selection of variants to be part of a specific platform instance. These restrictions and dependencies will be modelled in terms of constraints in the variability model in Section 4.2.1:

- 1) At least one content type must be present as otherwise the content-sharing platform is useless.
- 2) To ensure acceptable quality of service, the maximum bit rate for video content on the Tomcat web container is 128 kBit/s.
- 3) The combination of supported content types may be restricted based on the capabilities of the web container or the deployment platform, e.g. due to load problems only a limited number of content types may be available on the traditional deployment target.
- 4) Some content types may be served by a separate web container in order to configure a simple load balancing mechanism, for example 3D content should be served by a JBoss server. As a further extension, a web container may be configured to retrieve its content from a specific database.
- 5) Content types may be transformed and the result may be shared. Such transformations should be configured in terms of configuration chains, such as the textual representation of the audio track of a video. As transformations may be resource-consuming and, thus, affect the performance, on the traditional platform only simple and resource saving implementations should be deployed while resource-consuming high-quality transformations may be used on the cloud platforms.

This content-sharing platform product line will be developed in the following sections using EASy-Producer. In particular, we will focus on the variability modelling, the variability implementation and the derivation of a specific platform instance.

## **4.2. Defining a New Base Service Platform**

In this section, we will describe the process of defining the variability of a (base) service platform (a SPL) using EASy-Producer from the perspective of a domain engineer. We will start with the creation of a new product line project in EASy-Producer, define the configuration space in terms of an IVML variability model, and implement the variabilities using a variability implementation technique. Further, we will define a corresponding build script in VIL to specify the instantiation process of the variable artefacts. The resulting base service platform (the product line project) will be the basis for the derivation of different content-sharing platforms by an application engineer.

The first step towards a product line definition in EASy-Producer is to define a new product line project. For this purpose, start the Eclipse application with the already installed EASy-Producer tool (see Section 3 for installation details). Start the **New Project Wizard** by opening *File* → *New*

---

<sup>5</sup> Eucalyptus website: <http://open.eucalyptus.com/>

<sup>6</sup> Amazon cloud website: <http://aws.amazon.com/de/ec2/>

<sup>7</sup> Azure website: <http://www.microsoft.com/windowsazure/>

→ **Project**. Expand the EASy-Producer category and select the entry **New EASy-Producer Project**. This opens the **Product Line Project Wizard** that requires the definition of a name for the new product line project. In our example, we will use `PL_Content_Sharing` as the name of our prototypical product line. EASy suggests naming the newly created project with a prefix (`PL_`). However, it is not necessary to keep this prefix. Enter the name and click the *Finish* button. The product line project will be created and EASy-Producer will automatically open the **Product Line Editor** as illustrated in Figure 5.

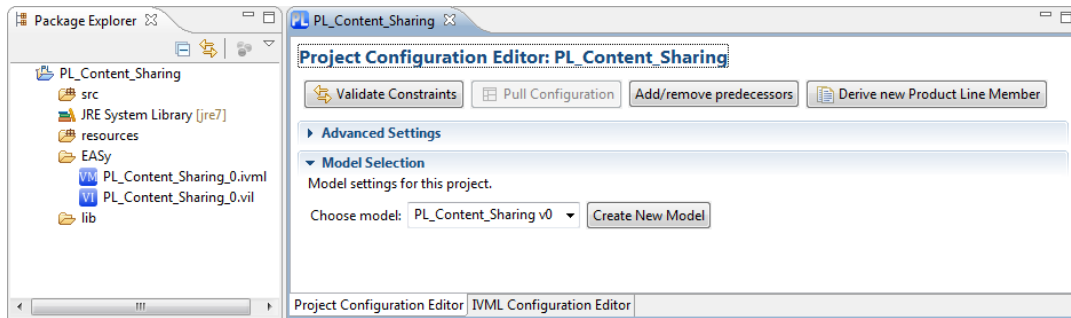


Figure 5: Running Example – The Product Line Editor.

The **Product Line Editor** is the central editor in EASy-Producer as it provides the basic information about a SPL (or a product) as well as the capabilities to derive, configure, and instantiate a product using the different tabs shown in Figure 5. For this purpose, the configuration space (variability model) and the implementation space (variability implementation) must be defined. We will describe both definitions in detail in the next two sections.

#### 4.2.1. Configuration Space Definition

A variability model defines the valid configuration space of a specific SPL. The variabilities are implemented in the artefacts. In EASy-Producer, we use IVML<sup>8</sup> for defining a variability model and, thus, the configuration space of the content-sharing platform. This model will be the basis for configuring individual service platforms in terms of defining valid value combinations for the configuration space elements (the IVML decision variables).

In EASy-Producer, each product line project comes with its own IVML-file, which can be opened and edited using the **IVML-Editor**. The IVML-file is located in the **EASy-folder** of the project. The name of the file is composed of the name of the product line and the version number (here initially “o”). In our example, double-click the file **PL\_Content\_Sharing\_o** in order to open the **IVML-Editor**.

By default, each IVML-file has a mandatory project element and a mandatory version number as shown in Figure 6. The project element is the top-level element of each IVML file and identifies the configuration space of a certain software project (product line or product). The version element defines the current state of evolution of a project and, thus, identifies a specific (state of a) project. The default version is “vo”.

<sup>8</sup> See the IVML language specification (cf. 3.4) for a detailed description of this language.



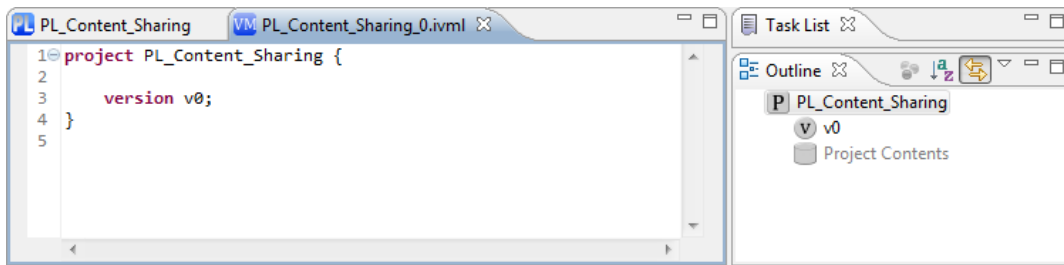


Figure 6: Running Example – The IVML Editor.

We characterize the configuration space of the variant-enabled content-sharing base platform by specifying the variability model in IVML. Figure 7 shows a snippet of the variability model (the complete model can be found in the appendix A.1). First, we define several enumerations that represent the different content types, container types, etc., which an application may support in general (lines 5-8). These enumerations are the basis for specifying the type, for example, of a specific content (lines 10-12). The basic content compound must be refined in order to represent the specific configuration options for Video, 3D (ThreeD), and BLOB contents (lines 14-27). The other compounds are modelled according to the running example (cf. Section 4.1). As indicated in the outline on the right side of Figure 7, the two types `Application` and `TargetPlatform` include decision variables of the previously defined (compound) types representing the complete set of configuration options for the content-sharing base platform. Thus, two variables (one of type `Application` and one of type `TargetPlatform`) are defined as the main decision variables for configuring a specific content-sharing platform variant. These variables will also be displayed in the **IVML Configuration Editor** tab of the **Product Line Editor**. We will discuss this editor in detail in the process of product configuration in Section 4.3.1.

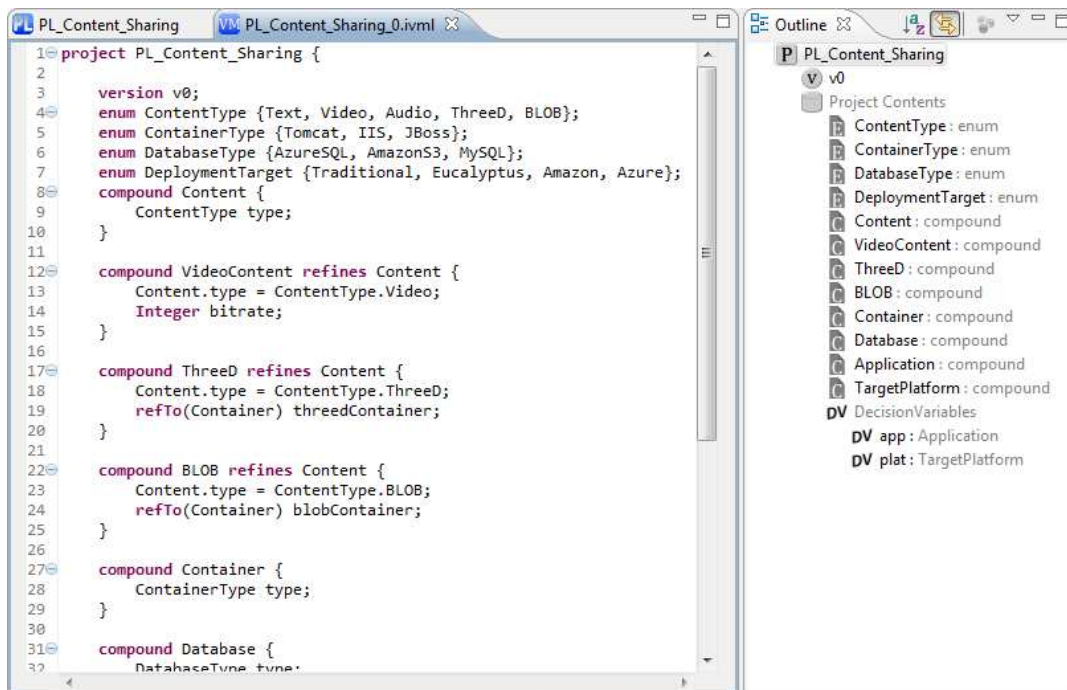


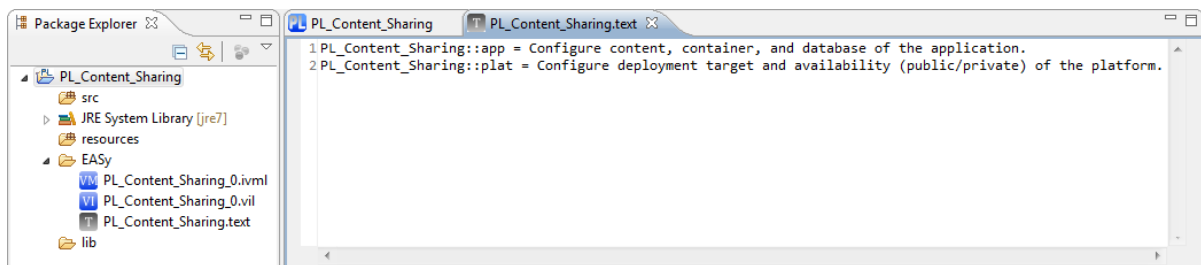
Figure 7: Running Example – The Variability Model (snippet).

In addition to the variability model, we will also define comments for the decision variables in order to support the application engineer in the configuration of a valid product. The first step is to create a new text-file in the **EASy-folder** of the product line project. For this purpose, right-click



on the **EASy-folder** and select **New** → **File**. The name of the file must match the name of the variability model (the project-name in IVML) followed by the version number to unambiguously link the comments to the desired decision variables. Thus, enter `PL_Content_Sharing_0.text` as the name of the new file and click the *Finish* button. We will open the new text-file with a simple text-editor to enter the comments. Each comment-definition will start with the name of the variability model (the project name) followed by “::”, the name of the decision variable for which we want to define the comment, and an equal-sign (“=”). The actual comment is defined in plain text after the “=”.

The result of the comment-definition is shown in Figure 8. We defined two comments for the two decision variables “app” and “plat” of the running example (cf. Figure 8a)), which are displayed in the **IVML Configuration Editor of the Product Line Editor** (cf. Figure 8b)).



a) Comment-definition in the text-file.

Filtering Options					
Decision Name	Current value	+	-	Freeze	Comment
app	UNDEFINED				Configure content, container, and database of the application.
plat	UNDEFINED				Configure deployment target and availability (public/private) of the platform.

b) Decision variables and comments in the IVML configuration editor

**Figure 8: Running Example - Definition of Decision Variable Comments.**

Finally, the variability model, and, thus, the configuration space of the content-sharing application is defined. We will use this model in Section 4.3.1 for configuring a specific content-sharing platform variant. However, in the next section we will first discuss the implementation of the variabilities. This includes the relation of the decision variables to the implementation in order to automatically instantiate different platform variants.

#### 4.2.2. Implementation Space Definition

The implementation space of a specific SPL represents all variable artefacts that can be instantiated according to a specific configuration. The actual implementation of these artefacts depends on the applied variability implementation techniques (VITs). A VIT is a specific approach to realize variability, e.g., using pre-processor directives, aspects, or any other techniques. In EASy-Producer different VITs can be applied and combined. Their application and combination is defined in a VIL<sup>9</sup> build script. However, some VITs may be realized by an individual instantiator, which actually applies the VIT. A detailed discussion on the concept of instantiators in EASy-Producer can be found in the EASy-Producer developers guide (cf. Section 3.4). In the running

<sup>9</sup> See the VIL language specification (cf. 3.4) for a detailed description of this language.

example, we will use the Velocity instantiator as it is one of the default instantiators of the basic EASy-Producer installation.

All product line (and product) source code is located in the **src** folder of the product line project as shown in Figure 9. The Velocity instantiator provides pre-processor functionality to Java and can be applied in terms of adding Velocity-specific statements to plain Java code. In lines 5 and 6 of Figure 9, the deployment platform and the public switch will be defined accordingly to the values of `platTarget` and the `isPublic` variables (cf. the variability model in Figure 7). Both variables are nested variables of the platform variable `plat`. Thus, they are accessed using “-”-notation. In order to guarantee that Velocity will find these variables, the instantiator requires a dollar-sign in front of the variable declarations in the code.



Figure 9: Running Example – The Variability Implementation (snippet).

The next step is to define the VIL build script. Open the **VIL Build Language Editor** by double clicking the **VIL-file** in the **EASy-folder**. The file has the name `PL_Content_Sharing_0.vil`. Figure 10 shows the VIL build script of this example (this build script can also be found in the appendix A.2). This script is rather simple: the first rule `clean` deletes all source artefacts in the target project (the product project) to guarantee that only the most recent instantiated artefacts are present in the final product. The second rule `main` is the entry-point for the VIL engine. This rule defines `clean` to be a precondition (guaranteeing that the target will be cleaned before the actual instantiation). The only action `main` defines is the call of the Velocity instantiator with the path to the generic source artefacts, the target path, and the configuration as parameters.

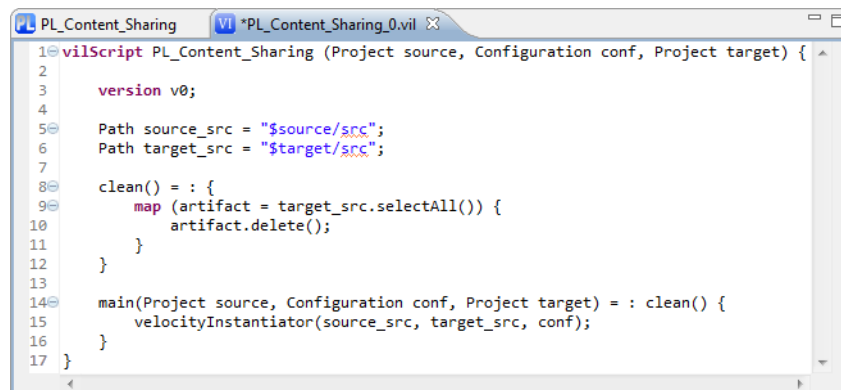


Figure 10: Running Example – The VIL Build Script.

Finally, the implementation space and the corresponding build script are defined to instantiate content-sharing application variants accordingly to a configuration. On this basis, we will derive a new product from this product line in the next section.

### 4.3. Deriving a Domain-Specific Service Platform

In this section, we will describe the process of deriving a new domain-specific platform from a software product line defined in EASy-Producer. We will adopt the perspective of an application engineer and start with the derivation of a new product line member<sup>10</sup> (in this case, the product project), configure the product based on the variability model defined in Section 4.2.1, and instantiate the product line artefacts accordingly. This will result in a specific content-sharing application variant with the desired functionalities ready for use.

The first step towards an instantiated domain-specific platform is to derive a new member from the previously defined base platform product line. For this purpose open the **Product Line Editor** by *right clicking* on the product line project and select *Edit Productline* in the context menu. In the **Project Configuration Editor** tab click the *Derive new Product Line Member* button, define a name for the new member, and click the *Ok* button. In our running example, we will use `Audio_Sharing_App` as the name of the new member. A new product line project will be created and the corresponding **Product Line Editor** will open automatically as shown in Figure 11.

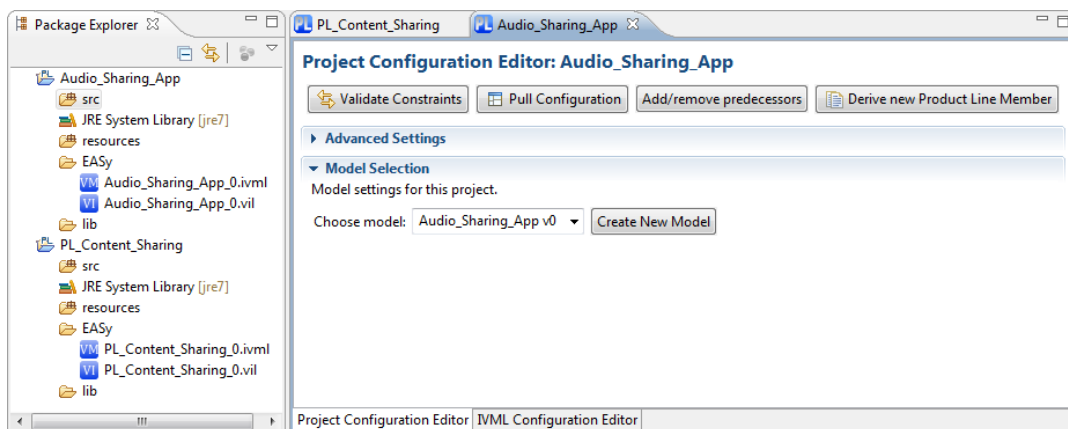


Figure 11: Running Example – The Product Derivation.

In the new product line member project, we will configure the desired functionalities of our specific audio content-sharing platform. This configuration will be used to finally instantiate the domain-specific platform. We will describe both steps in detail in the next two sections.

#### 4.3.1. Configuration of a Domain-Specific Service Platform

A product configuration (in this example the configuration of the domain-specific service platform) is a set of configured elements. In IVML configured elements are specified by assigning specific values to the elements in the configuration space, i.e. the decision variables, the attributes, etc. The validity of a configuration is checked against the constraints of the variability model using the built-in reasoning mechanism. The valid product configuration provides the basis for the (automated) instantiation of the corresponding product artefacts.

<sup>10</sup> In EASy-Producer, we do not distinguish between a product line infrastructure and a final product. Both are simply projects that may contain more or less variability (in case of a product none).

EASy-Producer provides two ways of configuring the elements of an IVML variability model: either use the **IVML Editor** by double-clicking the *IVML file* of the derived product line member (in our example the *Audio\_Sharing\_App\_0.ivml* file) in order to configure the elements of the imported project (the product line project) manually, or use the **IVML Configuration Editor** tab of the **Product Line Editor**. In our example, we will use the **IVML Configuration Editor**. This eases the configuration task as it includes all configurable elements of the imported project and provides the possible values for each of these elements automatically (we will discuss the configuration editor in detail in Section 5.2.2). Figure 12 illustrates the **IVML Configuration Editor**, including the configurable elements of our audio content-sharing application.

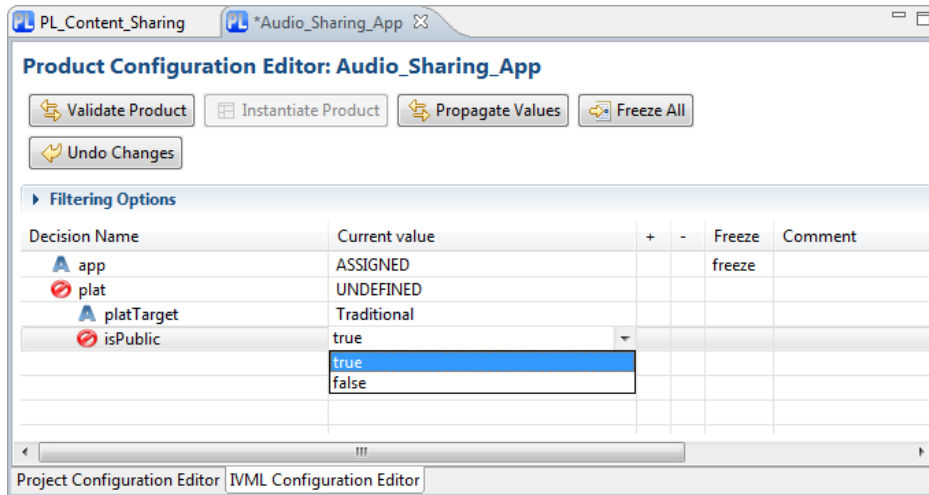


Figure 12: Running Example – The IVML Configuration Editor.

The next step is to check whether the configuration is valid. For this purpose, click on the *Validate Product* button of the **IVML Configuration Editor**. This executes the built-in IVML reasoning. If the product is valid, it is ready for instantiation. If it is not valid, the configuration must be revised in order to guarantee that the resulting product will work appropriately. In case of an invalid configuration, EASy-Producer will issue a description of the configuration problem and propose a possible error location in the current configuration.

Finally, the product is configured and ready for instantiation.

#### 4.3.2. Instantiation of a Domain-Specific Service Platform

Product instantiation describes the process of resolving the variability of product line artefacts according to a product configuration. This process results in the product artefacts that are mostly variation-free and ready to use. However, in some situations it is desired to resolve some of the variabilities at a later point in time, for example, at initialization time or runtime. In such a case, the instantiation process will leave these variabilities as-is.

EASy-Producer provides a fully automated instantiation process, which is based on the variability model, the current configuration and VIL build script. We defined this information in the previous sections, such as the implementation space and build script (cf. Section 4.2.2) and the product configuration (cf. Section 4.3.1). This relies in turn on the configuration space definition (cf. Section 4.2.1). Thus, the last step is to click the *Instantiate Product* button in the **IVML**

**Configuration Editor.** This will yield the instantiated artefacts from the product line project and inserts them into the product project while resolving the variabilities.

## 5. EASy-Producer in Detail

In this section, we will describe EASy-Producer in detail. This includes the description of the product line project structure in Section 5.1 as well as the different editors in Section 5.2.

### 5.1. The Product Line Project Structure

In this section, we will discuss the product line project structure of EASy-Producer. The basic structure of each product line project equals the general structure of Java-project in Eclipse. The only difference is in the EASy-folder of the product line project. This folder contains all EASy-Producer files. These files are:

File Icon	Description
-----------	-------------



The IVML-file, which contains the variability model described in the INDENICA Variability Modelling Language, or a specific configuration.  
This file is **mandatory** and will be automatically created if a new product line project is created.



The text-file, which contains additional comments for the decision variables defined in the variability model. Please note that we use a “T” for “Text” instead of a “C” for “Comments” as this may be confused with “Configuration”.  
This file is **optional** and has to be created manually.



The VIL build script file, which contains the specification of the instantiation process of the variable artefacts of the product line project.  
This file is **mandatory** and will be automatically created if a new product line project is created.



The VIL template file, which contains the definition of generic templates that can be applied during the instantiation process to create or manipulate specific artefacts and their content.  
This file is **optional** and has to be created manually.

All files introduced above can be created manually (also those that are mandatory) by clicking *File* → *New* → *Other*. In the wizard, open the **EASy-Producer**-folder and select the file you want to create. Please note that we recommend adding such files to the **EASy-folder** of the product line project as this is the default folder for EASy-specific files. Further, the creation of a variability model, a build script, and templates are supported by individual (text) editors that will open by simply double-clicking the respective file in the **EASy-folder**.

### 5.2. The Product Line Editor

The **Product Line Editor** is the central editor in EASy-Producer as it provides the basic information about a SPL (or a product) as well as the capabilities to derive, configure, and instantiate a product using the different sub-editors (tabs). This editor opens automatically if a new product line project is created. In order to open the editor manually, *right click* on the product line project and select **Edit Product Line**.

### 5.2.1. The Project Configuration Editor

The **Project Configuration Editor** provides the general configuration options of a product line project as well as the general actions that can be performed.

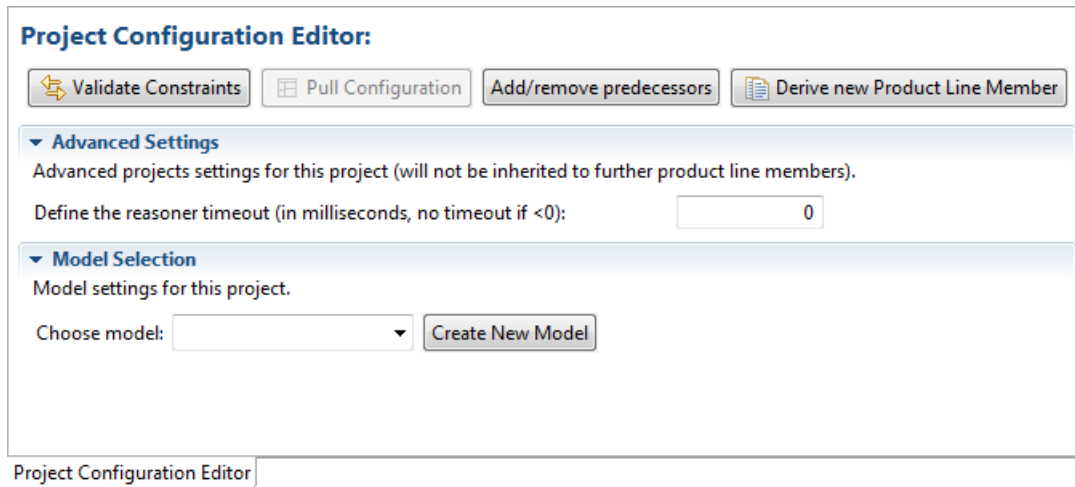


Figure 13: Project Configuration Editor.

Editor Element	Description
“Validate Constraint” Button	Validates all constraints in the selected variability model using the reasoner.
“Pull Configuration” Button	In case that a new predecessor is added (see below) the configuration of the predecessor is integrated into the configuration of the current product line project.
“Add/remove predecessor” Button	In a Multi Software Product Line scenario (cf. Section 2.3), a new product line project, e.g. a product line product, is initially derived from a single parent product line project, e.g. the base product line. In order to integrate additional parents, this button will open a dialog to select the desired product line projects of the current workspace in Eclipse. Please note that the addition of new parent product line projects also requires the Pull Configuration action (see above) to integrate the configurations of the selected parents into the current product line project.
“Derive new Product Line Member” Button	Derives a new product line project based on the current product line project. For example, this creates a new product project based on the base product line project.
Advanced Settings (Reasoner Timeout)	Restricts the time for the reasoner to calculate the validity of a specific configuration to the defined time in milliseconds.

## Model Selection

In case that multiple variability models are available, this option enables the selection of the desired variability model as the basis for the configuration of the current product line project.

### 5.2.2. The IVML Configuration Editor

The IVML Configuration Editor supports the configuration of individual products or partially configured product lines by providing a graphical user interface for the assignment of values to decision variables defined in the variability model. This editor is also used to start the instantiation process after the configuration of a specific product.

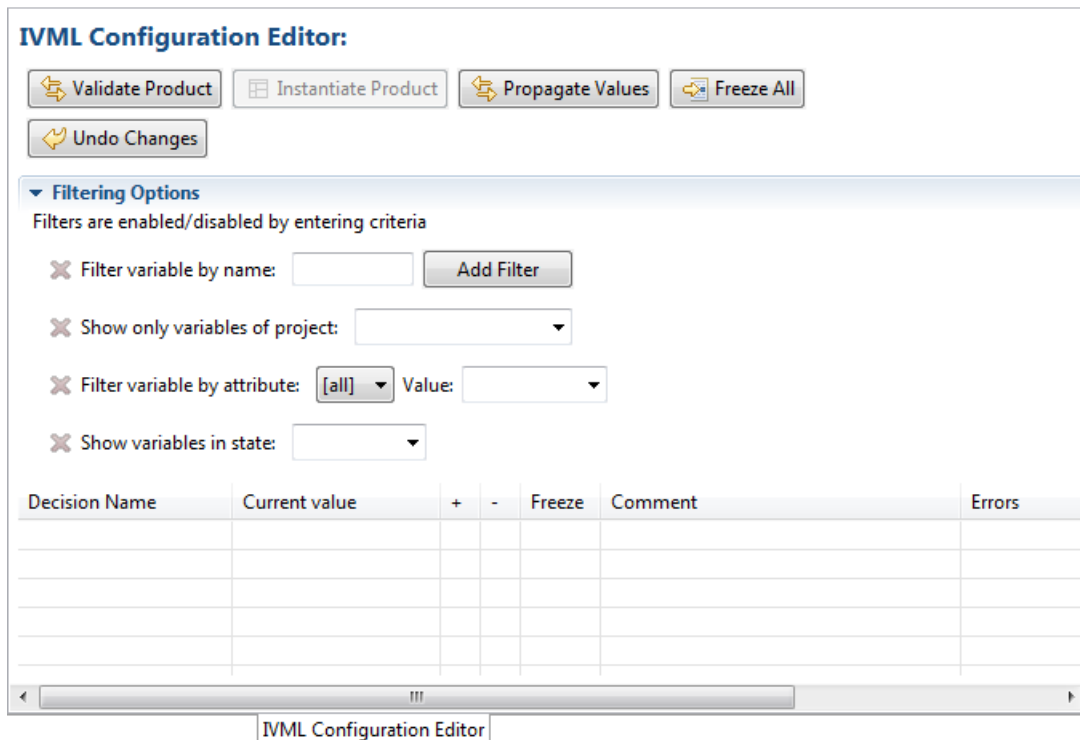


Figure 14: IVML Configuration Editor.

## Editor Element

## Description

“Validate Product” Button

Validates the current configuration of this product line project using the reasoner.

“Instantiated Product” Button

Instantiates the current project line project (partial product line or product) based on the current (partial) configuration.

“Propagate Values” Button

Assigns currently unassigned decision variables of the configuration automatically. This automation requires the assignment of a subset of the available decision variables and the relation of these variables to the unassigned variables in terms of constraints



“Freeze All” Button	in the variability model. Freezes all assigned decision variables of the current configuration. The values of frozen decision variables cannot be altered afterwards. For details on the concept of freezing, see the IVML language specification (cf. Section 3.4).
“Undo Changes” Button	Reverts all changes since the last saving of the current configuration.
Filtering Options (Filter by name)	Filters the available decision variables of the current configuration by name.
Filtering Options (Filter by project)	Filters the available decision variables of the current configuration by the project in which they are defined.
Filtering Options (Filter by attribute)	Filters the available decision variables of the current configuration by an attribute and the specific attribute value. For details on the concept of attributes, see the IVML language specification (cf. Section 3.4).
Filtering Options (Filter by state)	Filters the available decision variables of the configuration by their state. The available states are: unassigned, assigned, and frozen.
“Decision Name” Column	The name of the decision variable.
“Current value” Column	The current value of the decision variable.
“+” Column	Adds a new element to a sequence or a set of decision variables.
“-“ Column	Deletes an existing element from a sequence or a set of decision variables.
“Freeze” Column	Freezes the decision variable in the row of the current cell.
“Comment” Column	Displays additional information regarding the purpose and the configuration options of the decision variable. This requires the definition of a text-file (cf. Section 4.2.1).
“Errors” Column	Displays errors, for example the violation of a constraint of the variability model by the current value of the decision variable in that row. This requires the validation of the current configuration.

## 6. Frequently Asked Questions (FAQ)

In this section, we describe known problems and solutions while working with EASy.

### 6.1. VIL-Editor won't work after updating EASy/Xtext

#### Description

After an update of EASy or Xtext, the Editor for VIL scripts displays syntactic errors, even for valid VIL files. In Particular, the editor detects an error inside the first keyword of the file. This is usually the **vilScript** keyword. Also syntax highlighting and the content assist may not work completely. An example is given in Figure 15.

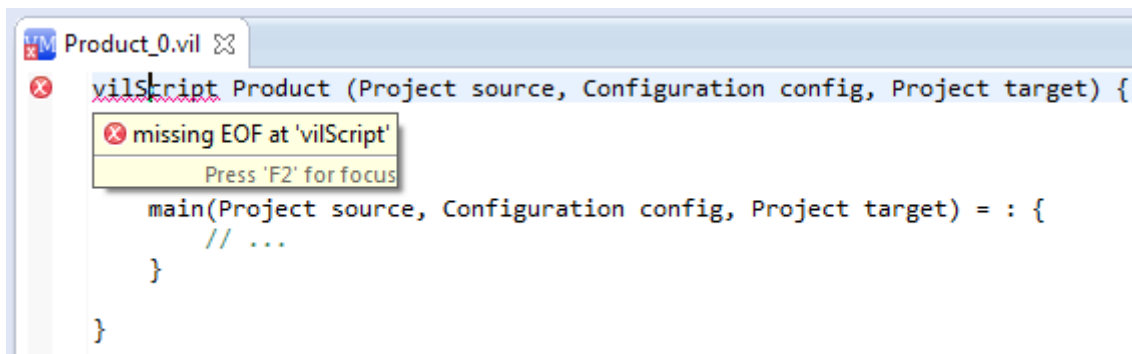


Figure 15: Broken VIL-Editor.

#### Cause

Eclipse stores information related to its plug-ins inside the workspaces. We observed that this information can cause problems when Xtext was updated.

#### Solution

Create a new (empty) workspace and import all existing projects into the new workspace using the Eclipse import function. This can be done as follows:

- Open in menu: *File* → *Import...*
- Select: *General* → *Existing Projects into Workspace*
- Select: *Select root directory*
- Click: *Browse*
- Select old workspace and confirm with OK
- Select Projects to import (we also suggest to check the option *Copy projects into workspace* (see Figure 16) ) and click on *Finish*

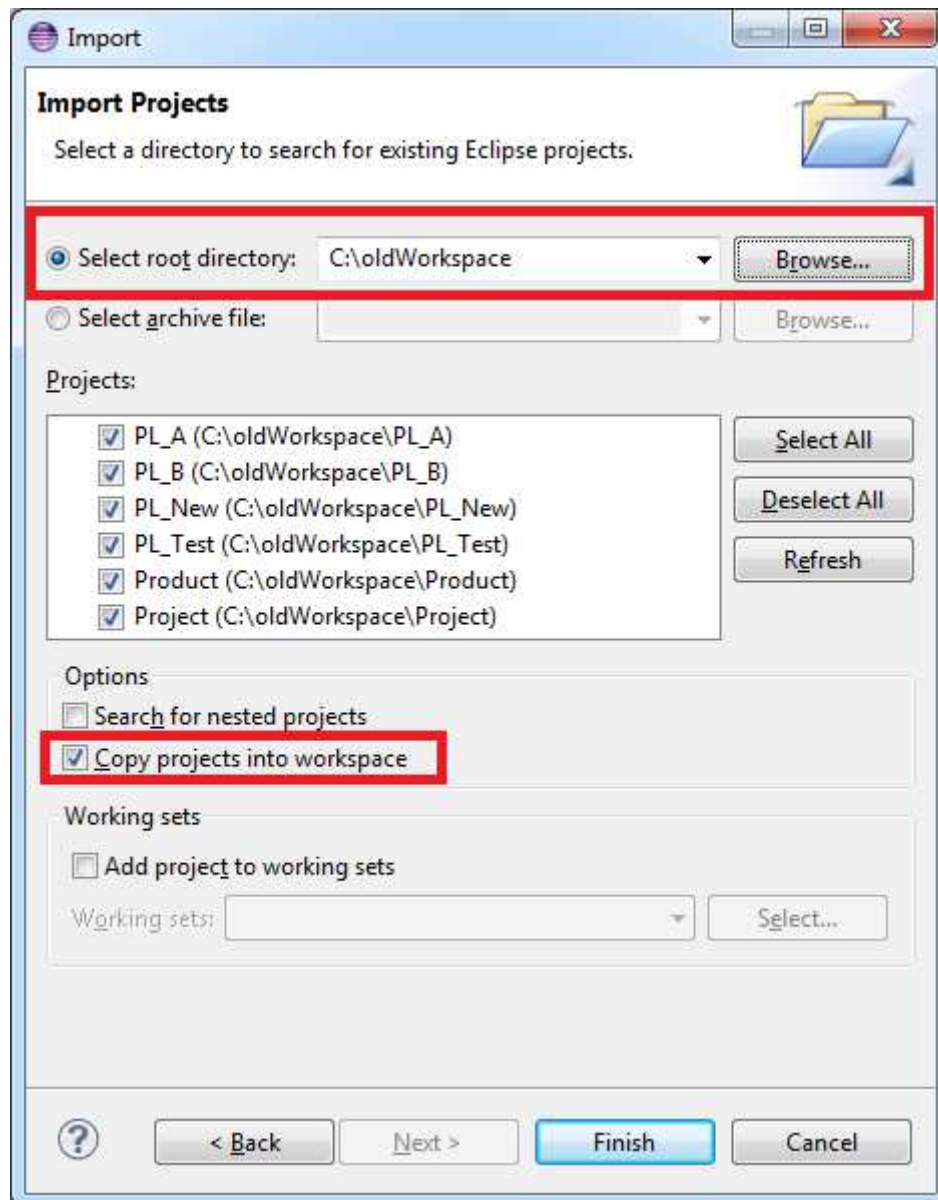


Figure 16: Importing old projects into a fresh workspace

## A. Appendix

### A.1. Running Example IVML-File

```
project PL_Content_Sharing {  
  
  version v0;  
  enum ContentType {Text, Video, Audio, ThreeD, BLOB};  
  enum ContainerType {Tomcat, IIS, JBoss};  
  enum DatabaseType {AzureSQL, AmazonS3, MySQL};  
  enum DeploymentTarget {Traditional, Eucalyptus, Amazon, Azure};  
  compound Content {  
    ContentType type = ContentType.Audio;  
  }  
  
  compound VideoContent refines Content {  
    Integer bitrate;  
    Content.type = ContentType.Video;  
  }  
  
  compound Container {  
    ContainerType type = ContainerType.Tomcat;  
  }  
  
  compound Database {  
    DatabaseType type = DatabaseType.MySQL;  
  }  
  
  compound Application {  
    Content appContent;  
    Container appContainer;  
    Database appDatabase;  
  }  
  
  compound TargetPlatform {  
    DeploymentTarget platTarget;  
    Boolean isPublic;  
  }  
  
  compound ThreeD refines Content {  
    refTo(Container) threedContainer;  
    Content.type = ContentType.ThreeD;  
  }  
  
  compound BLOB refines Content {  
    refTo(Container) blobContainer;  
    Content.type = ContentType.Audio;  
  }  
  
  Application app;  
  TargetPlatform plat;  
  
  plat.platTarget == DeploymentTarget.Traditional implies  
    app.appDatabase.type == DatabaseType.MySQL;  
  plat.platTarget == DeploymentTarget.Eucalyptus implies  
    app.appDatabase.type == DatabaseType.AmazonS3;
```

```
plat.platTarget == DeploymentTarget.Amazon implies
  app.appDatabase.type == DatabaseType.AmazonS3;
plat.platTarget == DeploymentTarget.Azure implies
  app.appDatabase.type == DatabaseType.AzureSQL;
}
```

## A.2. Running Example VIL Build Script-File

```
vilScript PL_Content_Sharing (Project source, Configuration conf, Project target)
{
  version v0;

  Path source_src = "$source/src";
  Path target_src = "$target/src";

  clean() = : {
    map (artifact = target_src.selectAll()) {
      artifact.delete();
    }
  }

  main(Project source, Configuration conf, Project target) = : clean() {
    velocityInstantiator(source_src, target_src, conf);
  }
}
```